UNIL | Université de Lausanne

A Dynamic Programming Code in C++

Simon Scheidegger
simon.scheidegger@unil.ch
August 30st, 2018

Cowles Foundation – Yale

Setup:

- We consider a infinite-horizon **discrete-state dynamic programming problem** similar to the example described in Judd (1998) – Numerical Methods in Economics, 12.1.3.

- Here, we have a stochastic growth problem with a stochastic component to the output.
  The rate of consumption is denoted by ***c***, and ***f (k,Θ)*** is the net-of-depreciation output when capital stock is ***k***, implying that capital evolves according to $\mathbf{k_{t+1} = k_t + f(k_t,\Theta_t) - c_t}$

- We assume a time-separable utility function, and $\boldsymbol{\beta < 1}$.

The optimal growth problem yields:

$$V(k,\Theta) = \max_{c_t} \mathbb{E} \sum_{t=0}^{\infty} \beta^t u(c_t)$$

$$\text{s.t.} \quad k_{t+1} = f(k,\Theta_t) - c_t$$

$$\Theta_{t+1} = g(\Theta_t)$$

Value function iteration

$$V_{new}(k,\Theta) = \max_{c} \left(u(c) + \beta \mathbb{E}\{V_{old}(k_{next},\Theta_{next})\}\right)$$

$$\text{s.t.} \quad k_{next} = f(k,\Theta_{next}) - c$$

$$\Theta_{next} = g(\Theta)$$

States of the model:

- k : today's capital stock
- Θ: today's productivity state

→ k, k_{next} , Θ and Θ_{next} are limited to a finite number of values

Specifications

Gross output:

$$f(k,\Theta) = k + \Theta(1 - \beta)k^{\alpha} /(\alpha\beta)$$

Utility function:

$$u(c) = c^{1-\gamma} /1-\gamma$$

choose $\gamma > 0$ in order for u(c) to be concave.

Parameters

$\alpha = 0.25$

$\beta = 0.9$

$\gamma = 5.0$

Range for capital stock: $\mathbf{[k_{min}, k_{max}] = [0.85, 1.15]}$

Discrete number of capital stocks: $\mathbf{n_k}$

Discretiztion of capital axis: kappa = $\mathbf{(k_{max} - k_{min})/(n_k - 1)}$

Productivity states: $\mathbf{n_\Theta = 5}$

$\mathbf{\Theta_{grid} = \{0.9, 0.95, 1.0, 1.05, 1.1\}}$

Transition Matrix: $\mathbf{\pi(i,j) = 1.0/n_\Theta = 1/5}$ (could be any other specification)

Computation

Since we solve a discrete-state DP problem,
the value function is a 2-dimensional matrix: **Val(k,Θ)**

→ We create **two Matrices Val_{old} and Val_{new}** and update them with value function iteration.

→ In our problem, the control is discrete. Hence, the maximization problem reduces to computing a list of values implied by all possible choices, and taking the maximum (no optimizer needed).

```
Do i = 1, n_Θ
  Do j = 1, kappa
    Val_new (i,j) = max(...)
  End do
End do
Val_old = Val_new
```

→ **Spot the huge potential for parallelization**

Setup of Code

1. cd ROOT/day1/code/DynamicProgramming/serial_DP

The folder contains the following important files:

main.cpp: the “driver routine” of the code.

solver.cpp: in this file, the actual value function iteration happens (→ parallelization).

econ.cpp: contains economic functions like utility, output.

parameters.cpp: specifies the parameters of the problem.

makefile: compiles the code

solver.cpp

```
for (int itheta=0; itheta<ntheta; itheta++) {

    /*
     Given the theta state, we now determine the new values and optimal policies corresponding to each
     capital state.
     */

    for (int ik=0; ik<nk; ik++) {

        // Compute the consumption quantities implied by each policy choice
        c=f(kgrid(ik), thetagrid(itheta))-kgrid;

        // Compute the list of values implied implied by each policy choice
        temp=util(c) + beta*ValOld*p(thetagrid(itheta));

        /* Take the max of temp and store its location.
         The max is the new value corresponding to (ik, itheta).
         The location corresponds to the index of the optimal policy choice in kgrid.
         */
        ValNew(ik, itheta)=temp.maxCoeff(&maxIndex);

        Policy(ik, itheta)=kgrid(maxIndex);

    }
}
```

loops to worry about

parameters.cpp

```
// Preference Parameters
double alpha = 0.25;
double gamm  = 5.0;
double beta  = 0.9;

// Capital Stock Interval
double kmin = 0.85; //minimum capital
double kmax = 1.15; //max. capital

// Choose nk, the number of capital stocks, and kappa, the discretization level
int nk  =  31;
double kappa  =  (kmax-kmin)/(nk-1.0);

// Number of theta states and the minimum and maximum theta values
int ntheta = 5;
double theta_min  = 1-10.0/100;
double theta_step = 5.0/100;

// Capital and Theta grids
ArrayXd kgrid=fill_kgrid();
ArrayXd thetagrid=fill_thetagrid();

// Number of Iterations
int numstart = 0; // start iterating at this timestep
int Numits = 3;   // stop  iterating at this timestep

// Maximum difference between succesive Value Function Iterates (initialized to 0)
double errmax = 0;

// Number of Columns for output
int nout = 3;

// Frequency with which data will be printed, e.g. 1=every timestep, 10=every ten iteration steps
int datafreq = 1;
```

Speeding-up the DP code

- We first parallelize the code with OpenMP (we look at solver.cpp).

OpenMP

- **We check the speed-up** for combinations of threads and a variety of **the discretization level** on one shared memory node.

MPI

- We parallelize the example with MPI (we look at solver.cpp).
- We check the speed-up for combinations of MPI processes and a variety of sample points (set in parameters.cpp).

- We parallelize the DP code in hybrid (mixed OpenMP/MPI).

Hybrid